1. Given the following declaration, write a snippet of C code that might lead to strlen(arr) returning no less than 8.

   char arr[4];
   strcpy(arr, "overflow"); or strcpy(arr, "overflow", 8);

2. Fill in the correct expression:

   char s1[MAX1];
   char s2[MAX2];
   getname(s2, MAX2); /* Initializes the string s2 */
   strncpy(s1, s2, MAX1 - (______));

3. a) Fill in the argument for malloc so that it allocates just enough space for the remaining code.

   char **s = malloc(3 * sizeof(char *));
   char p[10] = "Paul";
   char q[10] = "Karen";
   char r[10] = "Francois";

   *s = p;
   *(s+1) = q;
   *(s+2) = r;

   b) Write the above 3 statements using array notation so that they have the same effect.

      \[
      s[0] = p; \\
      s[1] = q; \\
      s[2] = r; \\
      \]

   c) Write one C statement to truncate the string "Francois" so that the following printf statement prints Fran

      printf("%s\n", r);

      \[
      r[9] = '\0'; \\
      \]

   d) Give the type of the following expressions. If the expression is not a pointer, also give its value.
4. Given the two declarations below circle the C statements that will compile without warning or error (other than those about unused variables):

```c
int *p;
int i = 10;
char q = i; char *c = p; double *f = &i; double d = i;
```

5. Show what is written to the file for each of the fprintf and fwrite statements. Show the value(s) in decimal and binary. ASCII values for characters: '0' is 48 (0x30), '1' is 49 (0x31), '6' is 54 (0x36)

```c
int i = 16;
fprintf(fp,"%d", i);
int j = 0x10;
fprintf(fp,"%d", j);
fwrite(&i, sizeof(int), 1,fp);
char c = i;
fwrite(&c, sizeof(char), 1,fp);
```